Refine Search

Search Results -

Term	Documents
(79 AND 78).USPT.	2
(L79 AND L78).USPT.	2

Database:

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L80

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Sunday, March 07, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; PLUR=YES; OP=ADJ		
L80	L79 and l78	2	L80
L79	call graph	163	L79
L78	L77 and l51	610	L78
L77	profil$ or performance monitor$	288071	L77
L76	L75 and l51	3	L76
L75	L74 and l52	325	L75
L74	invasive or introsive	24459	L74
L73	L72 and l52	4	L73
L72	code expansion	190	L72
L71	4937740.pn.	1	L71
L70	4845615.pn.	1	L70
L69	5193180.pn.	1	L69
L68	5193180.pn.	1	L68

L67	5164969.pn.	1	L67
L66	5142679.pn.	1	L66
L65	5047919.pn.	1	L65
L64	5047919.pn.	1	L64
L63	4937740.pn.	1	L63
L62	4845615.pn.	1	L62
L61	4845615.pn.	1	L61
L60	5963740.pn.	1	L60
L59	5828883.pn.	1	L59
L58	5732273.pn.	1	L58
L57	5539907.pn.	1	L57
L56	5539907.pn.	1	L56
L55	5465258.pn.	1	L55
L54	5450586.pn.	1	L54
L53	L52 and l47 and l51	3	L53
L52	profil$.ab. or performanc.ab.	27941	L52
L51	L50 same l49	3256	L51
L50	monitor$ near1 (code$1 or function$1)	13003	L50
L49	add$ or insert$	2739557	L49
L48	add near4 (code$1 or function$1)	13759	L48
L47	basic block	3346	L47
L46	5265254.pn.	1	L46
L45	5313616.pn.	1	L45
L44	5333304.pn.	1	L44
L43	5355487.pn.	1	L43
L42	5359533.pn.	1	L42
L41	5335344.pn.	1	L41
L40	5355484.pn.	1	L40
L39	5355487.pn.	1	L39
L38	5359533.pn.	1	L38
L37	5450586.pn.	1	L37
L36	5465258.pn.	1	L36
L35	5539907.pn.	1	L35
L34	5732273.pn.	1	L34
L33	5740443.pn.	1	L33
L32	5828883.pn.	1	L32
L31	5963740.pn.	1	L31
L30	6049666.pn.	1	L30
L29	5047919.pn.	1	L29
L28	5355487.pn.	1	L28
L27	5359533.pn.	1	L27

L26	5359533.pn.	1	L26
L25	5212794.pn.	1	L25
L24	5212794.pn.	1	L24
L23	5047919.pn.	1	L23
L22	4533997.pn.	1	L22
L21	4100532.pn.	1	L21
L20	5313616.pn.	1	L20
L19	L18 and l1	1	L19
L18	simulation	55439	L18
L17	L16 and l1	1	L17
L16	parallel	1173415	L16
L15	L14 and l1	1	L15
L14	parallel stack	584	L14
L13	L12 and l1	1	L13
L12	execut$	275754	L12
L11	L10 and l1	1	L11
L10	end code	2328	L10
L9	L8 and l1	1	L9
L8	exit	277720	L8
L7	L6 and l1	1	L7
L6	fixed number	14650	L6
L5	L4 and l3	1	L5
L4	l1 and l2	1	L4
L3	call graph	163	L3
L2	call chain	93	L2
L1	6126329.pn.	1	L1

END OF SEARCH HISTORY

ACM Digital Library

A half century of pioneering concepts and fundamental research have been digitized and indexed in a variety of ways in this special collection of works published by ACM since its inception. The ACM Digital Library includes bibliographic information, abstracts, reviews, and full texts.

Digital Library Overview
- **What's New**
- **FAQ**
- **DL Pearls**
- **Content and Organization**
- **Terms of Usage**
- **Resources from Affiliated Organizations**

Subscription and Access Information

> Access Information
> Individual Subscriptions
> Institutional Subscriptions

Browse the Digital Library

- **Journals**
- **Magazines**
- **Transactions**
- **Proceedings**
- **Newsletters**
- **Publications by Affiliated Organizations**

- **Special Interest Groups (SIGs)**

Personalized Services

My Bookshelf — Custom collections. Personal virtual Journals. Intelligent agent searches. Collaborative filtering.

Online Computing Reviews Service

OCRS — Access critical reviews of the computing literature using the Online Computing Reviews Service.

> Document Delivery Service

The ACM Digital Library is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

Read the ACM Privacy Policy and Code of Ethics
Questions? Comments? Contact webmaster@acm.org
Call: 1.800.342.6626 (USA & Canada) or +212.626.0500 (Global)
Write: ACM, 1515 Broadway, New York, NY 10036, USA

Try the *new* Portal design
Give us your opinion after using it.

Search Results

Search Results for: **[profiling and basic block and simulation and parallel]**
Found **320** of **127,944 searched.**

Warning: Maximum result set of 200 exceeded. Consider refining.

Search within Results

> Advanced Search

> Search Help/Tips

Sort by: Title Publication Publication Date Score

Results 1 - 20 of 200 short listing

Prev Page 1 2 3 4 5 6 7 8 9 10 Next Page

1 A framework for performance modeling and prediction 97%
Allan Snavely , Laura Carrington , Nicole Wolter , Jesus Labarta , Rosa Badia , Avi Purkayastha
Proceedings of the 2002 ACM/IEEE conference on Supercomputing November 2002

Cycle-accurate simulation is far too slow for modeling the expected performance of full parallel applications on large HPC systems. And just running an application on a system and observing wallclock time tells you nothing about why the application performs as it does (and is anyway impossible on yet-to-be-built systems). Here we present a framework for performance modeling and prediction that is faster than cycle-accurate simulation, more informative than simple benchmarking, and is shown usefu ...

2 Execution-driven simulation of multiprocessors: address and timing analysis 95%
S. Dwarkadas , J. R. Jump , J. B. Sinclair
ACM Transactions on Modeling and Computer Simulation (TOMACS) October 1994 Volume 4 Issue 4

This article describes and evaluates an efficient execution-driven technique for the simulation of multiprocessors that includes the simulation of system memory and that is driven by real program work loads. The technique produces correctly interleaved address traces at run-time without disk access overhead or hardware support, allowing accurate simulation of the effects of a variety of architectural alternatives on programs. We have implemented a simulator based on this technique that offe ...

3 Automatic performance prediction to support cross development of parallel programs 93%

Matthias Schumann
Proceedings of the SIGMETRICS symposium on Parallel and distributed tools
January 1996

4 Overlapping execution with transfer using non-strict execution for mobile programs 91%
Chandra Krintz , Brad Calder , Han Bok Lee , Benjamin G. Zorn
Proceedings of the eighth international conference on Architectural support for programming languages and operating systems October 1998
Volume 32 , 33 Issue 5 , 11

In order to execute a program on a remote computer, it mustfirst be transferred over a network. This transmission incurs the over-head of network latency before execution can begin. This latency can vary greatly depending upon the size of the program., where it is located (e.g., on a local network or across the Internet), and the bandwidth available to retrieve the program. Existing technologies, like Java, require that a jle be filly transferred before it can start executing. For large files an ...

5 Efficient simulation of multiprogramming 88%
W. P. Dawkins , V. Debbad , J. R. Jump , J. B. Sinclair
ACM SIGMETRICS Performance Evaluation Review , Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems
April 1990
Volume 18 Issue 1

6 Session 6B: Convergence of abstractions in high-level synthesis: Application-driven processor design exploration for power-performance trade-off analysis 87%
Diana Marculescu , Anoop Iyer
Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design November 2001

This paper presents an efficient design exploration environment for high-end core processors. The heart of the proposed design exploration framework is a two-level simulation engine that combines detailed simulation for critical portions of the code with fast profiling for the rest. Our two-level simulation methodology relies on the inherent clustered structure of application programs and is completely general and applicable to any microarchitectural power/performance simulation engine. The prop ...

7 Techniques for obtaining high performance in Java programs 87%
Iffat H. Kazi , Howard H. Chen , Berdenia Stanley , David J. Lilja
ACM Computing Surveys (CSUR) September 2000
Volume 32 Issue 3

This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portabili ...

8 System-level power optimization: techniques and tools 87%
Luca Benini , Giovanni de Micheli
ACM Transactions on Design Automation of Electronic Systems (TODAES) April

2000
Volume 5 Issue 2

This tutorial surveys design methods for energy-efficient system-level design. We consider electronic sytems consisting of a hardware platform and software layers. We consider the three major constituents of hardware that consume energy, namely computation, communication, and storage units, and we review methods of reducing their energy consumption. We also study models for analyzing the energy cost of software, and methods for energy-efficient software design and compilation. This survery ...

9 Application domains for fixed-length block structured architectures 87%

Lieven Eeckhout , Tom Vander Aa , Bart Goeman , Hans Vandierendonck , Rudy Lauwereins , Koen De Bosschere

Australian Computer Science Communications , Proceedings of the 6th Australasian conference on Computer systems architecture January 2001
Volume 23 Issue 4

In order to tackle the growing complexity and interconnects problem in modern microprocessor architectures, computer architects have come up with new architectural paradigms. A fixed-length block structured architecture (BSA) is one of these paradigms. The basic idea of a BSA is to generate blocks of instructions, called BSA-blocks, statically (by the compiler) and executing these blocks on a decentralized microarchitecture. In this paper, we focus on possible application domains for this archit ...

10 Shade: a fast instruction-set simulator for execution profiling 87%

Bob Cmelik , David Keppel

ACM SIGMETRICS Performance Evaluation Review , Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems May 1994
Volume 22 Issue 1

Tracing tools are used widely to help analyze, design, and tune both hardware and software systems. This paper describes a tool called Shade which combines efficient instruction-set simulation with a flexible, extensible trace generation capability. Efficiency is achieved by dynamically compiling and caching code to simulate and trace the application program. The user may control the extent of tracing in a variety of ways; arbitrarily detailed application state information may be collected ...

11 The rice parallel processing testbed 87%

R. C. Covington , S. Madala , V. Mehta , J. R. Jump , J. B. Sinclair

Proceedings of the 1988 ACM SIGMETRICS conference on Measurement and modeling of computer systems May 1988

12 Superscalar microarchitecture: Fetching instruction streams 85%

Alex Ramirez , Oliverio J. Santana , Josep L. Larriba-Pey , Mateo Valero

Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture November 2002

Fetch performance is a very important factor because it effectively limits the overall processor performance. However, there is little performance advantage in increasing front-end performance beyond what the back-end can consume. For each processor design, the target is to build the best possible fetch engine for the required performance level A fetch engine will be better if it provides better performance, but also if it takes fewer resources, requires less chip area, or consumes less power.In ...

13 Improving dynamic cluster assignment for clustered trace cache processors 85%

Ravi Bhargava , Lizy K. John
ACM SIGARCH Computer Architecture News , Proceedings of the 30th annual international symposium on Computer architecture May 2003
Volume 31 Issue 2

This work examines dynamic cluster assignment for a clustered trace cache processor (CTCP). Previously proposed cluster assignment techniques run into unique problems as issue width and cluster count increase. Realistic design conditions, such as variable data forwarding latencies between clusters and a heavily partitioned instruction window, increase the degree of difficulty for effective cluster assignment.In this work, the trace cache and fill unit are used to perform dynamic cluster assignme ...

14 Continuous profiling: where have all the cycles gone? 85%

Jennifer M. Anderson , Lance M. Berc , Jeffrey Dean , Sanjay Ghemawat , Monika R. Henzinger , Shun-Tak A. Leung , Richard L. Sites , Mark T. Vandevoorde , Carl A. Waldspurger , William E. Weihl
ACM SIGOPS Operating Systems Review , Proceedings of the sixteenth ACM symposium on Operating systems principles October 1997
Volume 31 Issue 5

15 Continuous profiling: where have all the cycles gone? 85%

Jennifer M. Anderson , Lance M. Berc , Jeffrey Dean , Sanjay Ghemawat , Monika R. Henzinger , Shun-Tak A. Leung , Richard L. Sites , Mark T. Vandevoorde , Carl A. Waldspurger , William E. Weihl
ACM Transactions on Computer Systems (TOCS) November 1997
Volume 15 Issue 4

This article describes the Digital Continuous Profiling Infrastructure, a sampling-based profiling system designed to run continuously on production systems. The system supports multiprocessors, works on unmodified executables, and collects profiles for entire systems, including user programs, shared libraries, and the operating system kernel. Samples are collected at a high rate (over 5200 samples/sec. per 333MHz processor), yet with low overhead (1–3% slowdown for most workloads). A ...

16 EEL: machine-independent executable editing 85%

James R. Larus , Eric Schnarr
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation June 1995
Volume 30 Issue 6

EEL (Executable Editing Library) is a library for building tools to analyze and modify an executable (compiled) program. The systems and languages communities have built many tools for error detection, fault isolation, architecture translation, performance measurement, simulation, and optimization using this approach of modifying executables. Currently, however, tools of this sort are difficult and time-consuming to write and are usually closely tied to a particular machine and operating sy ...

17 Limits of instruction-level parallelism 85%

David W. Wall
Proceedings of the fourth international conference on Architectural support for programming languages and operating systems April 1991
Volume 19 , 25 , 26 Issue 2 , Special Issue , 4

18 Exploiting dual data-memory banks in digital signal processors 85%
Mazen A. R. Saghir , Paul Chow , Corinna G. Lee
Proceedings of the seventh international conference on Architectural support for programming languages and operating systems September 1996
Volume 31 , 30 Issue 9 , 5

Over the past decade, digital signal processors (DSPs) have emerged as the processors of choice for implementing embedded applications in high-volume consumer products. Through their use of specialized hardware features and small chip areas, DSPs provide the high performance necessary for embedded applications at the low costs demanded by the high-volume consumer market. One feature commonly found in DSPs is the use of dual data-memory banks to double the memory system's bandwidth. When coupled ...

19 Compactly representing parallel program executions 85%
Ankit Goel , Abhik Roychoudhury , Tulika Mitra
ACM SIGPLAN Notices , Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming June 2003
Volume 38 Issue 10

Collecting a program's execution profile is important for many reasons: code optimization, memory layout, program debugging and program comprehension. Path based execution profiles are more detailed than count based execution profiles, since they present the *order* of execution of the various blocks in a program: modules, procedures, basic blocks etc. Recently, online string compression techniques have been employed for collecting compact representations of sequential program executions. I ...

20 Continuous program optimization: A case study 85%
Thomas Kistler , Michael Franz
ACM Transactions on Programming Languages and Systems (TOPLAS) July 2003
Volume 25 Issue 4

Much of the software in everyday operation is not making optimal use of the hardware on which it actually runs. Among the reasons for this discrepancy are hardware/software mismatches, modularization overheads introduced by software engineering considerations, and the inability of systems to adapt to users' behaviors.A solution to these problems is to delay code generation until load time. This is the earliest point at which a piece of software can be fine-tuned to the actual capabilities of the ...

Results 1 - 20 of 200 **short listing**

Prev Page 1 2 3 4 5 6 7 8 9 10 Next Page

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

ACM Digital Library

A half century of pioneering concepts and fundamental research have been digitized and indexed in a variety of ways in this special collection of works published by ACM since its inception. The ACM Digital Library includes bibliographic information, abstracts, reviews, and full texts.

Digital Library Overview
- **What's New**
- **FAQ**
- **DL Pearls**
- **Content and Organization**
- **Terms of Usage**
- **Resources from Affiliated Organizations**

Subscription and Access Information

> Access Information
> Individual Subscriptions
> Institutional Subscriptions

Browse the Digital Library

- **Journals**
- **Magazines**
- **Transactions**
- **Proceedings**
- **Newsletters**
- **Publications by Affiliated Organizations**

- **Special Interest Groups (SIGs)**

Personalized Services

My Bookshelf — Custom collections. Personal virtual Journals. Intelligent agent searches. Collaborative filtering.

Online Computing Reviews Service

OCRS — Access critical reviews of the computing literature using the Online Computing Reviews Service.

> Document Delivery Service

The ACM Digital Library is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

Read the ACM Privacy Policy and Code of Ethics
Questions? Comments? Contact webmaster@acm.org
Call: 1.800.342.6626 (USA & Canada) or +212.626.0500 (Global)
Write: ACM, 1515 Broadway, New York, NY 10036, USA

First Hit   Fwd Refs

End of Result Set

Generate Collection   Print

L23: Entry 1 of 1   File: USPT   Oct 3, 2000

US-PAT-NO: 6126329   issued by me   Majid

DOCUMENT-IDENTIFIER: US 6126329 A

TITLE: Method and apparatus for accurate profiling of computer programs

DATE-ISSUED: October 3, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bennett; James	Menlo Park	CA		
Anderson; Mark	Palo Alto	CA		
Na; Choon Piaw	Saratoga	CA		
Hastings; Reed	La Honda	CA		

07/970,315

07/718,573

US-CL-CURRENT: 717/127; 714/35, 717/130, 717/133

CLAIMS:

What is claimed is:

claim 1

1. A computer implemented method for profiling execution of a computer program having functions comprising instructions, the method comprising the steps of:

claim 10

for each function, parsing the function instructions into basic blocks, wherein a basic block consists of a contiguous sequence of at least one instruction of which the first instruction is a program control transfer destination, the last instruction is a program control transfer instruction, and any instructions between first and last instructions are neither program control transfer destinations nor program control transfer instructions; and   claim 11

for each basic block, performing under computer control, the steps of

claim 1 (i)

selecting a basic block;

analyzing the selected basic block to determine a fixed number of clock cycles for the selected basic block, and to determine whether the selected basic block contains an operating system (OS) call; and   claim 11

claim 1 (ii)

adding basic block profiling code so as to be executed whenever the basic block is executed, said basic block profiling code operating to add the determined fixed number of clock cycles to a clock cycle accumulator for the function, and further including, for at least one selected basic block containing an OS call, the OS call timing profiling steps of   claim 11

ii adding timing start code to determine a start time immediately before executing the OS call;

clm 1 (iii) adding timing end code to determine an end time immediately after executing the OS call;

adding OS call timing calculation code to determine from the start time and end time a number of clock cycles required to execute the OS call; and

adding OS call timing recording code to add the determined number of clock cycles to a clock cycle accumulator for the function;

clm 1 (b) wherein said OS call timing profiling steps are performed for substantially only basic blocks containing indeterminate-length-OS-calls.

2. A computer implemented method for profiling execution of a computer program having functions, the functions having instructions, the method comprising the steps of:

a) augmenting the program by adding profiling code to the program;

b) executing the augmented program and collecting profiling data for functions of the program, said profiling data including self+descendants times for the functions wherein the self+descendants times are percentages of a total execution time for the program; clm 4 % Total exec. time

c) constructing a call graph for the program; and

d) displaying to a user a representation of the call graph including arcs connecting nodes, wherein each arc links a parent function to a child

function and has a width corresponding to the self+descendants time for the child function, and wherein the step of displaying the representation of the call graph comprises displaying a call graph in which each arc has a width having a linear relationship to the percentage self+descendants time of the child function of the arc.

3. A computer implemented method for profiling execution of a computer program having the functions, the functions having instruction, the method comprising the steps of:

a) augmenting the program by adding profiling code to the program;

b) executing the augmented program and collecting profiling data for functions of the program, said profiling including self+descendants times for the functions wherein the self+descendants times are percentages of a total execution time for the program;

c) constructing a call graph for the program; and

d) displaying to a user a representation of the call graph including arcs connecting nodes, wherein each arc links a parent function to a child function and has a width corresponding to the self+descendants time for the child function and wherein the step of displaying the representation of the call graph comprise displaying a call graph in which each arc has a width having a logarithmic relationship to the percentage self+descendants time of the child function of the arc.

4. A computer implemented method for profiling execution of a computer program having functions, the functions having instructions, the method comprising the steps of:

a) augmenting the program by adding profiling code to the program;

b) executing the augmented program and collecting profiling data for functions of the program, said profiling data including self+descendants times for the functions;

c) constructing a call graph showing function calls for the program;

d) providing a pruning time filter value;

d) displaying to a user a representation of the call graph including arcs connecting nodes,

wherein each arc links a parent function to a child function, wherein a set of arcs is automatically pruned from said displayed call graph representation, wherein each of said set of said automatically pruned arcs connects to a child function having a self+descendants time less than said pruning time filter value.

5. The method of claim 4, wherein said step of providing the pruning time filter value comprises automatically determining, under computer control, a time value such that a predetermined number of the functions of the program have a self+descendants time at least equal to the determined time value.

6. The method of claim 5, wherein the predetermined number of functions is less than about one hundred.

7. The method of claim 6, wherein the predetermined number of functions is about thirty.

8. A computer implemented method for profiling execution of a computer program having functions comprising instructions, the method comprising the steps of:

for each function, parsing the function instructions into basic block, wherein a basic block consists of a contiguous sequence of at least one instruction of which the first instruction is a program control transfer destination, the last instruction is a program control transfer instruction, and any instructions between first and last instructions are neither program control transfer destinations nor program control transfer instructions; and

for each basic block, performing under computer control, the steps of

selecting a basic block;

analyzing the selected basic block to determine a fixed number of clock cycles for the selected basic block, and to determine whether the selected basic block contains an operating system (OS) call; and

adding basic block profiling code so as to be executed whenever the basic block is executed, said basic block profiling code operating to add the determined fixed number of clock cycles to a clock

cycle accumulator for the function, and further including, for at least one selected basic block containing an OS call, the OS call timing profiling steps of

adding timing start code to determine a start time immediately before executing the OS call;

adding timing end code to determine an end time immediately after executing the OS call;

adding OS call timing calculation code to determine from the start time and end time a number of clock cycles required to execute the OS call; and

adding OS call timing recording code to add the determined number of clock cycles to a clock cycle accumulator for the function;

wherein the added basic block profiling code operates to add the determined fixed number of clock cycles, and the added OS call timing recording code operates to add the determined number of cycles required to execute the OS call, to a clock cycle accumulator for the selected basic block of the function.

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 4845615 A

L70: Entry 1 of 1 File: USPT Jul 4, 1989

US-PAT-NO: 4845615
DOCUMENT-IDENTIFIER: US 4845615 A

TITLE: Software performance analyzer

DATE-ISSUED: July 4, 1989

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Blasciak; Andrew J.	Colorado Springs	CO		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Hewlett-Packard Company	Palo Alto	CA			02

APPL-NO: 07/ 106947 [PALM]
DATE FILED: October 14, 1987

PARENT-CASE:
This application is a continuation of application Ser. No. 605,515, filed Apr. 30, 1984, now abandoned.

INT-CL: [04] G06F 11/34

US-CL-ISSUED: 364/200; 364/265.6, 364/267.1, 364/267.3, 364/267.4, 364/300
US-CL-CURRENT: 714/10; 714/47

FIELD-OF-SEARCH: 364/2MSFile, 364/9MSFile, 364/300

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4080650	March 1978	Beckett	364/200
4231106	October 1980	Heap et al.	364/900
4321666	March 1982	Tasar et al.	364/200

4338660	July 1982	Kelley et al.	364/200
4382179	May 1983	Penton	377/20
4486827	December 1984	Shima et al.	364/200

ART-UNIT: 237

PRIMARY-EXAMINER: Shaw; Gareth D.

ASSISTANT-EXAMINER: Eakman; Christina M.

ATTY-AGENT-FIRM: Miller; Edward L.

ABSTRACT:

A software performance analyzer nonintrusively measures six different aspects of software execution. These include histograms or a table indicating the degree of memory activity within a collection of specified address ranges, or indicating the amount of memory of bus activity caused by the execution of programming fetched from within a collection of specified ranges, or indicating for a specified program the relative frequency with which it actually executes in specified lengths of time, or indicating for a specified program the relative frequency of a collection of specified available potential execution times (i.e., the complement of the previous measurement), or indicating for two specified programs the relative frequency of a specified collection time intervals between the end of one of the programs and the start of the other, or lastly, indicating the number of transitions between selected pairs of programs. All measurements may be either percentages relative to only the specified programs or ranges, or may be absolute percentages with respect to all activity occurring during the measurement. Acquired data may be in terms of time or of qualified occurrences of a specified event. Enable/disable and windowing for context recognition are available. The measurements are made by rendomly choosing and monitoring a first range for a selected period of time. An address range detector and bus status recognizer supply information to a state machine configured to control the particular type of measurement desired. Various counters are responsive to the state machine and accumulate data later reduced by software controlling the software performance analyzer. At the end of the monitoring period the next address range is monitored, and so on until the entire list has been used, whereupon a new random starting range is chosen and the measurement continues. The first two types of measurements listed above may also be performed in a real-time mode where two ranges are in fact monitored simultaneously and nearly continuously.

6 Claims, 19 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Term	Documents
"4845615"	1
4845615S	0
"4845615".PN..USPT.	1

(4845615.PN.).USPT.	1

Display Format: TI Change Format

Previous Page Next Page Go to Doc#

John Chavis
Eric Stamber

First Hit Fwd Refs
End of Result Set

Generate Collection Print

L29: Entry 2 of 2 File: USPT Apr 11, 2000

US-PAT-NO: 6049666
DOCUMENT-IDENTIFIER: US 6049666 A

TITLE: Method and apparatus for accurate profiling of computer programs

DATE-ISSUED: April 11, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bennett; James	Menlo Park	CA		
Anderson; Mark	Palo Alto	CA		
Na; Choon Piaw	Saratoga	CA		
Hastings; Reed	La Honda	CA		

US-CL-CURRENT: 717/130

CLAIMS:

What is claimed is:

1. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program;

pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions.

2. The method of claim 1, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value.

3. The method of claim 2, wherein the predetermined number of functions of the computer program is less than one hundred.

4. The method of claim 3, wherein the predetermined number of functions of the computer

h e b b g e e e f c e e ge

program is thirty.

5. The method of claim 1, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

6. The method of claim 5, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

7. The method of claim 1, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

8. The method of claim 1, further comprising augmenting the computer program to add profiling code.

9. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that prunes the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

10. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

11. The method of claim 10, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

12. The method of claim 10, further comprising pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value.

13. The method of claim 12, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value.

14. The method of claim 13, wherein the predetermined number of functions of the computer program is less than one hundred.

15. The method of claim 14, wherein the predetermined number of functions of the computer program is thirty.

16. The method of claim 10, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

17. The method of claim 10, further comprising augmenting the computer program to add profiling code.

18. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

19. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program;

pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of

functions of the computer program have self+descendants times equal to or greater than the pruning time filter value; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions.

20. The method of claim 19, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

21. The method of claim 20, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

22. The method of claim 19, wherein the predetermined number of functions of the computer program is less than one hundred.

23. The method of claim 22, wherein the predetermined number of functions of the computer program is thirty.

24. The method of claim 19, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

25. The method of claim 19, further comprising augmenting the computer program to add profiling code.

26. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that prunes the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

27. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including times for the functions;

constructing a call graph of function calls for the computer program; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative times for the functions.

28. The method of claim 27, wherein the graphical representations of the times for the functions are the widths of the arcs such that the widths are proportional to the times for the functions.

29. The method of claim 28, wherein the widths of the arcs are linearly or exponentially proportional to the times for the functions.

30. The method of claim 27, further comprising pruning the call graph to remove arcs depicting a function call to a function that has a time that is less than a pruning time filter value.

31. The method of claim 28, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have times equal to or greater than the pruning time filter value.

32. The method of claim 31, wherein the predetermined number of functions of the computer program is less than about one hundred.

33. The method of claim 32, wherein the predetermined number of functions of the computer program is about thirty.

34. The method of claim 27, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

35. The method of claim 27, further comprising augmenting the computer program to add profiling code.

36. The method of claim 27, wherein the times are self or self+descendants times for the functions.

37. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that displays a representation of the call graph including arcs that connect

functions to depict function calls and graphical representations that graphically approximate the relative times for the functions; and

a computer readable medium that stores said computer codes.

First Hit Fwd Refs
End of Result Set

Generate Collection Print

L29: Entry 2 of 2 File: USPT Apr 11, 2000

US-PAT-NO: 6049666
DOCUMENT-IDENTIFIER: US 6049666 A

TITLE: Method and apparatus for accurate profiling of computer programs

DATE-ISSUED: April 11, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bennett; James	Menlo Park	CA		
Anderson; Mark	Palo Alto	CA		
Na; Choon Piaw	Saratoga	CA		
Hastings; Reed	La Honda	CA		

US-CL-CURRENT: 717/130

CLAIMS:

What is claimed is:

1. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program;

pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions.

2. The method of claim 1, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value.

3. The method of claim 2, wherein the predetermined number of functions of the computer program is less than one hundred.

4. The method of claim 3, wherein the predetermined number of functions of the computer

program is thirty.

5. The method of claim 1, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

6. The method of claim 5, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

7. The method of claim 1, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

8. The method of claim 1, further comprising augmenting the computer program to add profiling code.

9. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that prunes the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

10. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

11. The method of claim 10, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

12. The method of claim 10, further comprising pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value.

13. The method of claim 12, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value.

14. The method of claim 13, wherein the predetermined number of functions of the computer program is less than one hundred.

15. The method of claim 14, wherein the predetermined number of functions of the computer program is thirty.

16. The method of claim 10, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

17. The method of claim 10, further comprising augmenting the computer program to add profiling code.

18. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

19. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

constructing a call graph of function calls for the computer program;

pruning the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of

functions of the computer program have self+descendants times equal to or greater than the pruning time filter value; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions.

20. The method of claim 19, wherein the graphical representations of the self+descendants times for the functions are the widths of the arcs such that the widths are proportional to the self+descendants times for the functions.

21. The method of claim 20, wherein the widths of the arcs are linearly or exponentially proportional to the self+descendants times for the functions.

22. The method of claim 19, wherein the predetermined number of functions of the computer program is less than one hundred.

23. The method of claim 22, wherein the predetermined number of functions of the computer program is thirty.

24. The method of claim 19, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

25. The method of claim 19, further comprising augmenting the computer program to add profiling code.

26. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including self+descendants times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that prunes the call graph to remove arcs depicting a function call to a function that has a self+descendants time that is less than a pruning time filter value, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have self+descendants times equal to or greater than the pruning time filter value;

computer code that displays a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative self+descendants times for the functions; and

a computer readable medium that stores said computer codes.

27. A method for profiling execution of a computer program including functions, comprising:

collecting profiling data for the functions during execution of the computer program, the profiling data including times for the functions;

constructing a call graph of function calls for the computer program; and

displaying a representation of the call graph including arcs that connect functions to depict function calls and graphical representations that graphically show the relative times for the functions.

28. The method of claim 27, wherein the graphical representations of the times for the functions are the widths of the arcs such that the widths are proportional to the times for the functions.

29. The method of claim 28, wherein the widths of the arcs are linearly or exponentially proportional to the times for the functions.

30. The method of claim 27, further comprising pruning the call graph to remove arcs depicting a function call to a function that has a time that is less than a pruning time filter value.

31. The method of claim 28, wherein the pruning time filter value is automatically determined under computer control so that a predetermined number of functions of the computer program have times equal to or greater than the pruning time filter value.

32. The method of claim 31, wherein the predetermined number of functions of the computer program is less than about one hundred.

33. The method of claim 32, wherein the predetermined number of functions of the computer program is about thirty.

34. The method of claim 27, wherein collecting profiling data includes determining a time for an operating system call by recording a start time before the operating system call and recording an end time after the operating system call such that the time for the operating system call is the difference between the start and stop times.

35. The method of claim 27, further comprising augmenting the computer program to add profiling code.

36. The method of claim 27, wherein the times are self or self+descendants times for the functions.

37. A computer program product for profiling execution of a computer program including functions, comprising:

computer code that collects profiling data for the functions during execution of the computer program, the profiling data including times for the functions;

computer code that constructs a call graph of function calls for the computer program;

computer code that displays a representation of the call graph including arcs that connect

functions to depict function calls and graphical representations that graphically approximate the relative times for the functions; and

a computer readable medium that stores said computer codes.

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 6049666 A

L30: Entry 1 of 1 File: USPT Apr 11, 2000

US-PAT-NO: 6049666
DOCUMENT-IDENTIFIER: US 6049666 A

TITLE: Method and apparatus for accurate profiling of computer programs

DATE-ISSUED: April 11, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bennett; James	Menlo Park	CA		
Anderson; Mark	Palo Alto	CA		
Na; Choon Piaw	Saratoga	CA		
Hastings; Reed	La Honda	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Rational Software Corp.	Santa Clara	CA			02

APPL-NO: 09/ 138111 [PALM]
DATE FILED: August 21, 1998

PARENT-CASE:
CROSS REFERENCE TO RELATED APPLICATIONS This application is a continuation of Ser. No. 08/871,247, filed Jun. 9, 1997; which is a continuation of Ser. No. 08/074,428, now abandoned, filed Jun. 8, 1993. This application is related to U.S. Ser. No. 07/970,315, filed on Nov. 2, 1992 now U.S. Pat. No. 5,335,344, which is a continuation of U.S. Ser. No. 07/718,573, filed on Jun. 21, 1991, now U.S. Pat. No. 5,193,180.

INT-CL: [07] G06 F 9/44

US-CL-ISSUED: 395/704
US-CL-CURRENT: 717/130

FIELD-OF-SEARCH: 395/704

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4845615	July 1989	Blasciak	364/200
4937740	June 1990	Agarwal et al.	364/200
5047919	September 1991	Sterling et al.	364/200
5142679	August 1992	Owaki et al.	395/700
5164969	November 1992	Alley et al.	
5193180	March 1993	Hastings	395/704
5212794	May 1993	Pettis et al.	395/700
5247651	September 1993	Clarisse	395/500
5265254	November 1993	Blasciak et al.	395/704
5313616	May 1994	Cline et al.	395/500
5333304	July 1994	Christensen et al.	395/575
5335344	August 1994	Hastings	395/704
5355484	October 1994	Record et al.	395/704
5355487	October 1994	Kellar et al.	395/650
5359533	October 1994	Ricka et al.	364/484
5450586	September 1995	Kuzara et al.	395/704
5465258	November 1995	Adams	395/704
5539907	July 1996	Srivastava et al.	395/705
5732273	March 1998	Srivastava et al.	395/704
5740443	April 1998	Carini	395/705
5828883	October 1998	Hall	395/704
5963740	October 1999	Srivastava et al.	395/704

OTHER PUBLICATIONS

Graham et al., GPROF; A Call Graph Execution Profiler, p. 120-126 (1-16) 1982.
Hilfingen, A Memory Allocation Profiler for C and Lisp Programs, p. 223-37, 1988.
Ponder et al., Inaccuracies in Program Profilers, p. 459-467, 1987.
Graham et al., Executions Profilers for modular Programs.
Michael D. Smith, "Tracing with Pixie," Apr. 4, 1991, Stanford University Technical Report No. CSL-TR-91-497, pp. 1-29.
"Pixie", UNIX man pp. 1-2.
James R. Larus et al., "Re-writing Executable Files to Measure Program Behavior", Mar. 25, 1992, Univ. of Wisconsin, Computer Science Department.
Thomas Ball, "Optimally Profiling and Tracing Programs", Sep. 6, 1991, Univ. of Wisconsin, Computer Science Department.

ART-UNIT: 272

PRIMARY-EXAMINER: Stamber; Eric W.

ASSISTANT-EXAMINER: Chavis; John Q.

ATTY-AGENT-FIRM: Ritter, Van Pelt & Yi, LLP

ABSTRACT:

An object code expansion profiler equips a program for execution profiling by preprocessing the object code files of the program so as to add profiling

monitoring code to the beginning of all or substantially all functions. The preprocessing includes, for each function, the steps of grouping the function's instructions into basic blocks, counting the number of cycles required to execute the instructions of the basic block, and inserting special monitoring code with the basic block. The special monitoring code is executed each time the basic block is executed, and updates the profiling information to reflect the number of cycles required to execute the basic block. Special handling is provided for profiling calls to the Operating System (OS). The resultant profiling information is converted into a call graph image most useful for human users. For each arc in the graph connecting a calling-function/parent-node to a called-function/child node, the displayed arc image has a width logarithmically proportional to the self+descendants time for the called function.

37 Claims, 25 Drawing figures

Term	Documents
"6049666"	1
6049666S	0
"6049666".PN..USPT.	1
(6049666.PN.).USPT.	1

h e b b g e e e f e f ef b e